

# DEEP LEARNING WITH SETS AND POINT CLOUDS

**Siamak Ravanbakhsh, Jeff Schneider & Barnabás Póczos**

School of Computer Science

Carnegie Mellon University

Pittsburgh, PA 15213, USA

{mravanba, jeff.schneider, bapoczos}@cs.cmu.edu

## ABSTRACT

We study a simple notion of structural invariance that readily suggests a parameter-sharing scheme in deep neural networks. In particular, we define structure as a collection of relations, and derive graph convolution and recurrent neural networks as special cases. We study composition of basic structures in defining models that are invariant to more complex “product” structures such as graph of graphs, sets of images or sequence of sets. For demonstration, our experimental results are focused on the setting where the discrete structure of interest is a set. We present results on several novel and non-trivial problems on sets, including point-cloud classification, set outlier detection and semi-supervised learning using clustering information.

## 1 INTRODUCTION

The holy grail of representation learning is to discover and exploit the invariances of the data. Performing this task within the deep learning framework (LeCun et al., 2015) can benefit from its immense capacity and trainability. This has motivated many recent works on finding the invariances in data or defining models with known invariances.

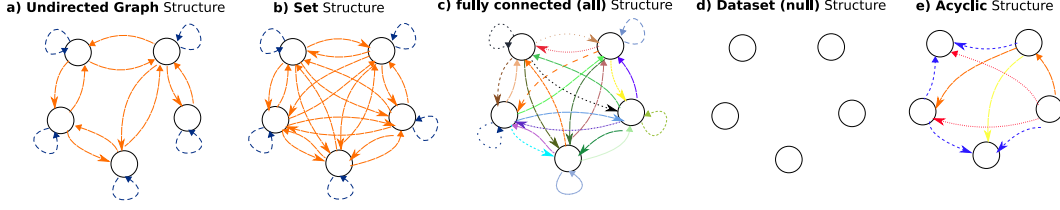
In the following Section 2 defines a minimal notion of structure and invariance of a function with respect to this structure. We then consider an abstract neuron as our function class of interest and derive some of the familiar convolution layers using this formulation. Section 3 studies composition of new structures using two definitions of graph product and pooling. Section 4 defines a partial ordering on structures with the data and feature dimension at extremes of this ordering. This partial ordering also suggests an incremental approach for handling product structures. Appendix B briefly studies acyclic structures, where under some conditions, a forward pass in the deep network performs ancestral sampling in a deep directed graphical model, with recurrent networks as a special case.

Section 5 studies application of these ideas to the “set” structure, one of the most basic structures with a wide range of applications that have so far remained unexplored in the context of deep learning. In particular, we consider application of set-invariant layers in detecting set outliers, and performing point-cloud classification where particles are treated as set members. Application of set-invariant layer in a semi-supervised setting is discussed in Appendix C. Section 6 reviews the related work and puts our contribution in perspective.

## 2 STRUCTURE AND INVARIANCE

Given a set  $\mathcal{X} = \{x_1, \dots, x_n, \dots, x_N\}$ , with  $x_n \in \mathfrak{R}$ , a (binary) relation  $\mathbb{I}_{\mathcal{X}} \subseteq \mathcal{X} \times \mathcal{X}$  is a collection of ordered pairs  $(x, x')$  of members of  $\mathcal{X}$ . We define the structure  $\mathbb{S}_{\mathcal{X}} = \{\mathbb{I}_{\mathcal{X}}^1, \dots, \mathbb{I}_{\mathcal{X}}^R \mid \mathbb{I}_{\mathcal{X}}^r \cap \mathbb{I}_{\mathcal{X}}^{r'} = \emptyset, 1 \leq r, r' \leq R\}$ , as a collection of non-overlapping relations on  $\mathcal{X}$ . We will drop the subscript  $\mathcal{X}$  whenever the set  $\mathcal{X}$  is evident from the context. We will use this minimal notion of structure to define simple rules of parameter-sharing that respect invariances of the structure.

For this purpose we need to define functions on the set  $\mathcal{X}$ . We use the subscript  $x_{\mathbb{I}} = \{x' \mid (x', x) \in \mathbb{I}\}$  to denote dependencies of  $x$  in  $\mathbb{I}$ . Note that  $x_{\mathbb{I}}$  is a set and our abuse of notation is for simplicity.

Figure 1: Colored di-graph representation of different structures  $\mathbb{S}$ .

The structural dependencies of  $x$  in  $\mathbb{S}$  are accordingly defined as the union of its relational dependencies  $x_{\mathbb{S}} = \bigcup_{\mathbb{I} \in \mathbb{S}} x_{\mathbb{I}}$ . We are interested in the functions  $f$  associated with  $x \in \mathcal{X}$ . These functions are defined over the domain of each  $x_{\mathbb{S}}$  – that is for each  $x \in \mathcal{X}$ ,  $f : \mathbb{R}^{|x_{\mathbb{S}}|} \rightarrow \mathbb{R}$ . Here,  $x_{\mathbb{I}} \subseteq \mathcal{X}$  has no particular ordering information.

Let the tuple  $\vec{\mathcal{X}}$  denote  $\mathcal{X}$  with some fixed ordering. Sub-tuples  $\vec{x}_{\mathbb{I}}$  and  $\vec{x}_{\mathbb{S}}$  inherit the same ordering.  $S^x$  denotes the symmetric group of all permutations of size  $|x_{\mathbb{S}}|$ . The action  $\pi(\vec{x}_{\mathbb{S}})$  of each group member  $\pi \in S^x$  is a re-ordering of  $\vec{x}_{\mathbb{S}}$ . Each  $x_{\mathbb{I}} \subseteq x_{\mathbb{S}}$  identifies a sub-group  $S_{\mathbb{I}}^x \leq S^x$  consisting of all permutations of elements in  $x_{\mathbb{I}}$ . We say a function  $f$  is invariant to relation  $\mathbb{I}$  iff  $f(\vec{x}_{\mathbb{S}}) = f(\pi(\vec{x}_{\mathbb{S}}))$  for  $\pi \in S_{\mathbb{I}}^x$  and it is invariant to the structure  $\mathbb{S}$  if the same identity holds for  $\pi \in S_{\mathbb{S}}^x = S_{\mathbb{I}_1}^x \dots S_{\mathbb{I}_r}^x \dots S_{\mathbb{I}_R}^x$  – i.e., it is invariant to all its relations  $\mathbb{I} \in \mathbb{S}$ . Since we often work with functions  $f$  that are invariant to  $\mathbb{S}$ , in the rest of the paper we will use  $f(x_{\mathbb{S}})$  to denote  $f(\vec{x}_{\mathbb{S}})$ .

We consider a sub-class of functions in the following form

$$f_{\theta}(x_{\mathbb{S}}) = \sigma \left( \bigoplus_{\mathbb{I} \in \mathbb{S}, x \in x_{\mathbb{I}}} \theta_{\mathbb{I}} x \right) \quad (1)$$

where  $\oplus$  is an associative and commutative operation such as maximization or summation,  $\theta_{\mathbb{I}} \in \mathbb{R}$  are parameters shared within a relation and  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  is for example a sigmoid. Note that the total number of parameters required by functions in  $\mathcal{F} = \{f_{\theta} : \mathbb{R}^{|x_{\mathbb{S}}|} \rightarrow \mathbb{R} \mid x \in \mathcal{X}\}$  depends only on  $|\mathbb{S}|$  rather than  $|\mathcal{X}|$ .

**Proposition 2.1.**  $f_{\theta} \in \mathcal{F}$  is invariant to  $\mathbb{S}$ .<sup>1</sup>

Our definition of structure  $\mathbb{S}$  can be visualized as an edge-colored directed graph (di-graph), where each color identifies a relation  $\mathbb{I}$  over the vertex-set  $\mathcal{X}$ . We use this representation in our examples; see Fig. 1.

**Example 2.2.** Figure 2(a:left) shows the colored di-graph representation of  $\mathbb{S}$  for a single input/output channel in a **2D convolution** with  $3 \times 3$  kernel.  $\mathbb{S}$  consists of 9 relations corresponding to 9 styles in connections of Fig. 2(a:left). Here,  $x_{\mathbb{I}}$  for each variable contains a single member and  $x_{\mathbb{S}}$  is the collection of these 9 relations. If we wished our convolution operation to be invariant to up-down and left-right flip, we could share the corresponding parameters, in which case  $|\mathbb{S}| = 4$ .

**Example 2.3.** In **graph convolution** all graph-edges are considered equivalent. Here, the structure contains two types of relation relations  $\mathbb{S} = \{(x, x') \mid (x, x') \in \mathcal{E}\}, \{(x, x) \mid x \in \mathcal{X}\}$ , where  $\mathcal{E}$  is the edge-set. Figure 1(a) shows the structure for an undirected graph. Using  $\oplus = \text{mean}$ , the parameter sharing function of Eq. (1) becomes  $f_{\bar{\theta}, \dot{\theta}}(x_{\mathbb{S}}) = \sigma \left( \frac{\bar{\theta}}{|x_{\mathbb{S}}|} \sum_{x' \in x_{\mathbb{S}}} x' + \dot{\theta} x \right)$ . If we assume an ordering on  $\mathcal{X}$  to get  $\vec{\mathcal{X}}$ , we can rewrite this expression for  $\vec{\mathcal{X}}$  as  $f_{\bar{\theta}, \dot{\theta}}(\vec{\mathcal{X}}) = \sigma(\bar{\theta}(\tilde{\mathbf{A}}\vec{\mathcal{X}}) + \dot{\theta}(\mathbf{I}\vec{\mathcal{X}}))$  where  $\tilde{\mathbf{A}} = \mathbf{A} \mathbf{D}^{-1}$  is the normalized binary adjacency matrix,  $\mathbf{I}$  is the identity matrix and  $\bar{\theta}, \dot{\theta} \in \mathbb{R}$ . Here  $f$  and  $\sigma$  are applied element-wise. This is similar in form (aside from normalization) to the graph-convolution in (Kipf & Welling, 2016), with single input/output channels. To see how multiple channels affect the structure, we should first define composition of structures.

### 3 INVARIANCES OF STRUCTURAL COMPOSITION

Deep learning models seldom work with a single structure. As we will see in this section, even the classic multi-layer perceptron assumes a composite structure. We consider special forms of

<sup>1</sup>See Appendix for proofs.

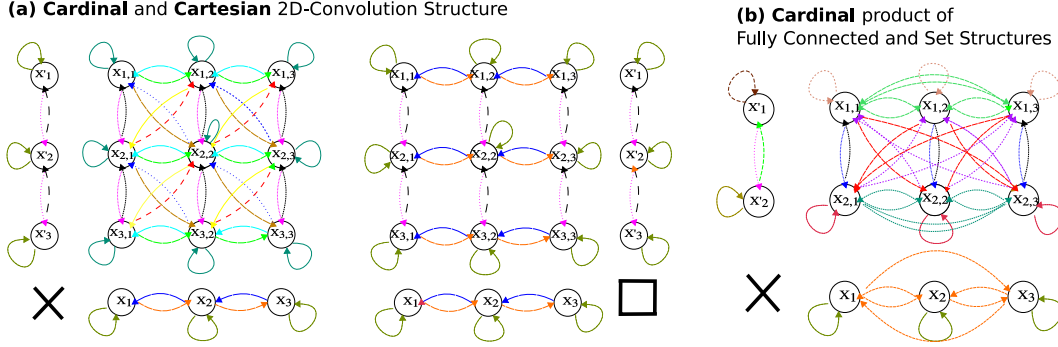


Figure 2: **a)** variations of 2D-convolution (grid) as different structural product, **b)** Cardinal product of set and fully connected structure defines multiple output channels for the set-invariant layer.

composition, where different structures are defined on “dimensions” of the set  $\mathcal{X}$ , where the notion of dimension comes from the Cartesian product. More specifically, we start with sets  $\mathcal{X}^1, \dots, \mathcal{X}^d$ , each having their own structure  $\mathbb{S}_{\mathcal{X}^1}, \dots, \mathbb{S}_{\mathcal{X}^d}$  and define a new structure on the product set  $\mathcal{X}^1 \times \dots \times \mathcal{X}^d$ . Examples of such compositions are when we have a data-set of feature-vectors, sequence of sets or a graph of graphs. Here, we introduce two families of composition:

**Cardinal Composition.** Let  $\mathbb{S}_{\mathcal{X}^1}$  and  $\mathbb{S}_{\mathcal{X}^2}$  be structures over the set  $\mathcal{X}^1$  with  $|\mathcal{X}^1| = N_1$  and  $\mathcal{X}^2$  with  $|\mathcal{X}^2| = N_2$  respectively. Their cardinal composition  $\mathbb{S}_{\mathcal{X}^1} \times \mathbb{S}_{\mathcal{X}^2}$  defines a new structure over the Cartesian product set  $\mathcal{X}^{1,2} = \mathcal{X}^1 \times \mathcal{X}^2 = \{x_{1,1}, x_{1,2}, \dots, x_{1,N_1}, x_{2,1}, \dots, x_{N_1,N_2}\}$ . For this we first define the *product of two relations* as the tensor-product of their di-graph representation:

$$\mathbb{I}_{\mathcal{X}^1} \times \mathbb{I}_{\mathcal{X}^2} \stackrel{\text{def}}{=} \{(x_{i,i'}, x_{j,j'}) \mid (x_i, x_j) \in \mathbb{I}_{\mathcal{X}^1} \wedge (x_{i'}, x_{j'}) \in \mathbb{I}_{\mathcal{X}^2}\}.$$

We then define the Cardinal (or Tensor) product of two structures as the product of all pairs of their relations:  $\mathbb{S}_{\mathcal{X}^1} \times \mathbb{S}_{\mathcal{X}^2} \stackrel{\text{def}}{=} \{\mathbb{I}_{\mathcal{X}^1} \times \mathbb{I}_{\mathcal{X}^2} \mid \mathbb{I}_{\mathcal{X}^1} \in \mathbb{S}_{\mathcal{X}^1} \wedge \mathbb{I}_{\mathcal{X}^2} \in \mathbb{S}_{\mathcal{X}^2}\}.$

**Cartesian Composition.** We first define the *extension* of relation  $\mathbb{I}_{\mathcal{X}^1}$  to the Cartesian product domain  $\mathcal{X}^1 \times \mathcal{X}^2$

$$\mathbb{I}_{\mathcal{X}^1 \rightarrow \mathcal{X}^1 \times \mathcal{X}^2} \stackrel{\text{def}}{=} \{(x_{i,i'}, x_{j,j'}) \mid (x_i, x_j) \in \mathbb{I}_{\mathcal{X}^1} \wedge x_{i'} \in \mathcal{X}^2\}$$

where we define new pairs for all the combinations of pairs in  $\mathbb{I}_{\mathcal{X}^1}$  and members of the new domain  $\mathcal{X}^2$ .

Given  $\mathbb{S}_{\mathcal{X}^1}$  and  $\mathbb{S}_{\mathcal{X}^2}$ , the cardinal composition  $\mathbb{S}_{\mathcal{X}^1} \square \mathbb{S}_{\mathcal{X}^2}$  is defined as the union of extension of their member relations

$$\mathbb{S}_{\mathcal{X}^1} \square \mathbb{S}_{\mathcal{X}^2} \stackrel{\text{def}}{=} \{\mathbb{I}_{\mathcal{X}^1 \rightarrow \mathcal{X}^1 \times \mathcal{X}^2} \mid \mathbb{I}_{\mathcal{X}^1} \in \mathbb{S}_{\mathcal{X}^1}\} \cup \{\mathbb{I}_{\mathcal{X}^2 \rightarrow \mathcal{X}^1 \times \mathcal{X}^2} \mid \mathbb{I}_{\mathcal{X}^2} \in \mathbb{S}_{\mathcal{X}^2}\}.$$

**Example 3.1.** A structure that is often present is the **dataset structure**. Multiple instances in the dataset can be thought of as the product of a dataset structure,  $\mathbb{S}_{\mathcal{X}}^{\text{null}} = \emptyset$  with other structures. It is easy to check that this product, be it Cartesian or Cardinal, simply replicates the rest of structure for each data-instance and the corresponding parameter-sharing scheme shares all the model parameters across the dataset, where the number of parameters does not grow with the size of the dataset.

The opposite extreme of structure is the **unstructured data features** that assume  $\mathbb{S}_{\mathcal{X}}^{\text{all}} = \{\{(x, x')\} \mid \forall x, x' \in \mathcal{X}\}$ , and the resulting parameter-sharing scheme, the fully connected layer, shares no parameter.

**Example 3.2.** To highlight the difference between Cartesian and Cardinal composition in a familiar setting, let us proceed with the example of 2D convolution as the product of 1D convolutions. Here, the structures  $\mathbb{S}_{\mathcal{X}^1} = \{\mathbb{I}^{\text{left}}, \mathbb{I}^{\text{center}}, \mathbb{I}^{\text{right}}\}$  and  $\mathbb{S}_{\mathcal{X}^2} = \{\mathbb{I}^{\text{up}}, \mathbb{I}^{\text{center}}, \mathbb{I}^{\text{down}}\}$ , each have 3 relations as shown in Fig. 2(a). The figure then compares the resulting Cartesian and Cardinal product of  $\mathbb{S}_{\mathcal{X}^1}$  and  $\mathbb{S}_{\mathcal{X}^2}$ .



Figure 3: Each row shows a set, constructed from CelebA dataset, such that all set members except for an outlier, share at least two attributes (on the right). The outlier is identified with a red frame. The model is trained by observing examples of sets and their anomalous members, *without access to the attributes*. The probability assigned to each member by the outlier detection network is visualized using a red bar at the bottom of each image. The probabilities in each row sum to one. See Appendix D for more examples.

It is evident that the cardinal product defines relations based on all combination of original relations, while the Cartesian product, considers their union – therefore, it often reduces the total number of relations and increases parameter-sharing.

The multiplicity of channels/feature-maps, when for example using convolution, can be studied at both layers below and above. From the lower layer’s perspective, we are simply defining multiple functions that are invariant to the same structure. From the point of view of the layer above, however, multiple *channels* with the same structure are often considered unstructured, similar to the fully connected layer – i.e.,  $\mathbb{S}_{\mathcal{X}}^{\text{all}} = \{\{(x, x')\} \mid \forall x, x' \in \mathcal{X}\}$ . This structure is then composed with the main structure using the Cardinal product.

**Example 3.3.** Figure 2(b) shows the structure of the output of a set-invariant layer with set-size of 3 that uses 2 output channel. The resulting structure has  $2 \times 2 = 4$  parameters (2 parameters of the set-invariant layer is multiplied by the number of channels since we are using Cardinal product).

### 3.1 POOLING

To denote layer  $\ell \in \{0, \dots, L\}$ , we use super-script in parentheses. Using this notation

$$\mathcal{X}^{(\ell)} = \{f_{\theta^{(\ell)}}(x_{\mathbb{S}^{(\ell-1)}}^{(\ell-1)}) \mid x^{(\ell-1)} \in \mathcal{X}^{(\ell-1)}\} \quad \forall 1 \leq \ell \leq L \quad \text{and} \quad \mathcal{X}^{(0)} = \mathcal{X}$$

represents the sets of variables at layer  $\ell$  as a function of previous layer, where  $\mathcal{X}^{(0)} = \mathcal{X}$  is the input. We define pooling as the application of a commutative and associative binary operation such as maximization, averaging, product or summation over  $\mathcal{P} \subseteq 2^{\mathcal{X}^{(\ell)}}$ :

$$\mathcal{X}_{\mathcal{P}}^{(\ell)} \stackrel{\text{def}}{=} \left\{ \bigoplus_{x^{(\ell)} \in \mathcal{P}} x^{(\ell)} \mid \mathcal{P} \in \mathcal{P} \right\}.$$

Since  $\oplus$  is invariant to the order of its summands, for  $x^{(\ell)}, y^{(\ell)} \in \mathcal{X}^{(\ell)}$ , both invariant to  $\mathbb{S}^{(\ell-1)}$ ,  $x^{(\ell)} \oplus y^{(\ell)}$  is invariant to  $\mathbb{S}^{(\ell-1)}$  as well. It follows that *Pooling*  $\mathcal{X}^{(\ell)} \rightarrow \mathcal{X}_{\mathcal{P}}^{(\ell)}$  preserves invariances of  $\mathcal{X}^{(\ell)}$  with respect to the structure  $\mathbb{S}^{(\ell-1)}$  of that layer.

The commonly used max-pooling in images and cluster-pooling (a.k.a. coarsening) for graph structure fit within this definition. A useful partitioning of pooling variables are the ones that are consistent with a product structure. Let  $\mathcal{X} = \{x_{1,1}, \dots, x_{1,N_1}, \dots, x_{N_2,1}, \dots, x_{N_2,N_1}\}$  be the product of  $\mathcal{X}^1$  and  $\mathcal{X}^2$ . Partitioning of  $\mathcal{X}$  wrt  $\mathcal{X}^1$  is defined as  $\mathcal{X} \setminus \mathcal{X}^1 = \{\{x_{1,1}, \dots, x_{N_2,1}\}, \dots, \{x_{N_1,1}, \dots, x_{N_2,N_1}\}\}$  – that is we have one set per each member of  $\mathcal{X}^1$  and the structure  $\mathbb{S}_{\mathcal{X}^1}$  is preserved. This type of pooling is specially useful in semi-supervised learning, when the output of the network retains a structure  $\mathbb{S}_{\mathcal{X}^1}^1$ , while pooling over other structures.

## 4 A PARTIAL ORDERING OF STRUCTURES

In practical settings, it may be technically challenging to build network layers that perform weight-sharing for invariance wrt a complex product structure. Ideally, we want to reuse weight-sharing layers that are invariant to individual elements of the composition, and at the time retain invariance to the product structure.

However, maintaining invariance is often not enough – that is we need the parameter-sharing scheme to be sensitive (not invariant) to permutations over  $\vec{\mathcal{X}}$  that are not induced by the given structure  $\mathbb{S}$ . We call such functions (or equivalently, parameter-sharing scheme) *minimally invariant* to  $\mathbb{S}$ . The functions of the form Eq. (1) are minimally invariant to  $\mathbb{S}$ .

We formalize this notion using a partial ordering of all structures. We define a partial ordering, such that if the function  $f(\mathcal{X})$  is invariant to  $\mathbb{S}'$ , it is also invariant to  $\mathbb{S} \leq \mathbb{S}'$ . We say  $\mathbb{S}' \geq \mathbb{S}$  iff we can construct  $\mathbb{S}'$  starting from  $\mathbb{S}$  and using arbitrary repetition of the following operations (that strictly increase invariances of  $\mathbb{S}$ )

1. Merging of  $\mathbb{I}_{\mathcal{X}}^1, \mathbb{I}_{\mathcal{X}}^2 \Rightarrow \mathbb{I}_{\mathcal{X}}^1 \cup \mathbb{I}_{\mathcal{X}}^2$
2. Shrinking a relation  $\mathbb{I}_{\mathcal{X}}^1 \Rightarrow \mathbb{I}_{\mathcal{X}}^1 \setminus \mathbb{I}'_{\mathcal{X}}$ , which removes members of some  $\mathbb{I}'_{\mathcal{X}} \subseteq \mathbb{I}_{\mathcal{X}}$ .

**Example 4.1.** A simple inspection of Fig. 1(c,d) wrt the operations above shows that dataset structure  $\mathbb{S}^{\text{null}}$  and fully connected structure  $\mathbb{S}^{\text{all}}$  are the **greatest and the least elements of our partial ordering** – that is  $\mathbb{S}_{\mathcal{X}}^{\text{all}} \leq \mathbb{S}_{\mathcal{X}} \leq \mathbb{S}_{\mathcal{X}}^{\text{null}} \forall \mathbb{S}_{\mathcal{X}}$ . Therefore, in a sense our exploitation of structure in deep models appears in between two dimensions of data in a traditional multi-layer perceptron.

Now we extend ordering of structures to their product.

**Claim 4.2.** Composition of structures preserves their partial ordering

$$\mathbb{S}_{\mathcal{X}}^1 \leq \mathbb{S}_{\mathcal{X}}^2 \Rightarrow \mathbb{S}_{\mathcal{X}}^1 \times \mathbb{S}_{\mathcal{X}'} \leq \mathbb{S}_{\mathcal{X}}^2 \times \mathbb{S}_{\mathcal{X}'} \quad \wedge \quad \mathbb{S}_{\mathcal{X}}^1 \square \mathbb{S}_{\mathcal{X}'} \leq \mathbb{S}_{\mathcal{X}}^2 \square \mathbb{S}_{\mathcal{X}'} \quad \forall \mathbb{S}_{\mathcal{X}'} \quad (2)$$

### 4.1 ONE STRUCTURE AT A TIME

From our definition of partial ordering, it follows that a function  $f$  is minimally invariant to  $\mathbb{S}$  if it is not invariant to any  $\mathbb{S}' > \mathbb{S}$ . Due to “partial” ordering,  $f$  could be minimally invariant to several distinct structures. A sensible approach to handling one structure at a time, is to maintain invariance in all layers, while ensuring that at least one layer is minimally invariant to each component of the product structure.

The key to maintaining invariance (which is not minimal) is to use claim 4.2: Since  $\mathbb{S}_{\mathcal{X}^2}^{\text{null}} \geq \mathbb{S}_{\mathcal{X}^2} \forall \mathbb{S}_{\mathcal{X}^2}$ , from the claim 4.2 it follows that any function  $f$  that is invariant to  $\mathbb{S}_{\mathcal{X}^1}^{\text{null}} \times \mathbb{S}_{\mathcal{X}^2}^{\text{null}} \times \mathbb{S}_{\mathcal{X}^3}$  is also invariant to  $\mathbb{S}_{\mathcal{X}^1}^{\text{null}} \times \mathbb{S}_{\mathcal{X}^2} \times \mathbb{S}_{\mathcal{X}^3}$ . Since  $\mathbb{S}_{\mathcal{X}^1}^{\text{null}} \times \mathbb{S}_{\mathcal{X}^2}^{\text{null}} = \mathbb{S}_{\mathcal{X}^1 \times \mathcal{X}^2}^{\text{null}}$ , the implication is that we can treat a subset of structural dimensions the same way that we handle dataset, while maintaining invariance to the ignored structure. However, this comes at the cost, demonstrated in the following example.

**Example 4.3.** Given a decomposition of  $\mathcal{X}$  to  $\mathcal{X}^1 \times \mathcal{X}^2$ , the fully connected structure can be decomposed accordingly  $\mathbb{S}_{\mathcal{X}}^{\text{all}} = \mathbb{S}_{\mathcal{X}^1}^{\text{all}} \times \mathbb{S}_{\mathcal{X}^2}^{\text{all}}$ . By handling each of these structures at their own layer, we maintain the invariances that are provided by  $\mathbb{S}_{\mathcal{X}}^{\text{all}}$  (which is basically no invariance). However, our model does not capture the inter-dependence between the variables in  $\mathcal{X}^1$  and  $\mathcal{X}^2$ . This suggests caution when using this shortcut. Our use of this trick in the experiments is minimal – i.e., we use this to handle the convolution/grid structure, without worrying about the “set” structure at first few layers of our model for face outlier detection.

## 5 SET STRUCTURE

In this paper, we only focus our experimental results on the set data-structure. This simple structure alone is applicable in many settings including distribution regression and distribution classification which have become popular recently (Szabo et al., 2016). They have proven to be very useful in many applied problems from computer vision (Poczos et al., 2012) via neuroscience (Oliva et al., 2014) and robotics (Tallavajhula et al., 2016) to cosmology (Ntampaka et al., 2016). Our approach to



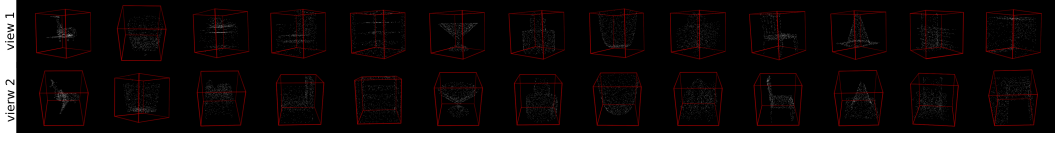


Figure 4: Examples for 13 out of 40 object classes in the ModelNet40. Each point-cloud is produced by sampling 1000 particles from the mesh representation of the original ModelNet40 instances. Top and the bottom row simply show different views of the same point-cloud.

(semi-)supervised learning with sets also generalizes various settings in multi-instance learning (Ray et al., 2011; Zhou et al., 2009).

In the following, after introducing the set-invariant layer in Section 5.1, we explore several novel applications. In particular for the vision task we perform outlier detection on CelebA face dataset in Section 5.2. Section 5.3 studies an important application of sets in representing low-dimensional point-clouds. We show that deep networks can successfully classify objects using their point-cloud representation. Appendix C studies application of semi-supervised learning with set structure in cosmology for prediction of galaxy red-shift using galaxy-clustering information.

### 5.1 PARAMETER-SHARING

Figure 1(b) shows an example of a set data-structure  $\mathbb{S}_{\mathcal{X}}^{\text{set}} = \{\{(x, x') \mid \forall x, x' \in \mathcal{X}\}, \{(x, x) \mid \forall x \in \mathcal{X}\}\}$ . The corresponding weight-sharing scheme applied to Eq. (1) has exactly two parameters<sup>2</sup>

$$f_{\bar{\theta}, \dot{\theta}}(x_{\mathbb{S}}) = \sigma\left(\bigoplus_{x' \in \mathcal{X}, x' \neq x} \bar{\theta}x'\right) \hat{\oplus} \dot{\theta}x \quad (3)$$

Here,  $x_{\mathbb{S}} = \mathcal{X}$  and  $f$  is associated with a particular member  $x \in \mathcal{X}$ . The two parameters  $(\dot{\theta}, \bar{\theta})$  of  $f$  account for two relations in  $\mathbb{S}^{\text{set}}$ .

Let us assume we have  $K$  input channels, with a set of size  $N$  and  $K'$  output channels. Using the matrix form  $\vec{\mathcal{X}} \in \mathbb{R}^{N \times K}$  for the input,  $\bar{\theta}, \dot{\theta} \in \mathbb{R}^{K \times K'}$  as parameters, and assuming  $\oplus = \text{mean}$  and  $\hat{\oplus} = +$ , the output is  $\vec{\mathcal{X}}' = \sigma(\vec{\mathcal{X}} \dot{\theta} + \frac{1}{N} \mathbf{1} \mathbf{1}^T \vec{\mathcal{X}} \bar{\theta})$ , where  $\mathbf{1}$  is the unit column vector of size  $N$ .

We found using  $\oplus = -\max$  to be slightly advantageous in some settings. Ignoring the  $x' \neq x$  in Eq. (3), this choice of operations gives  $\vec{\mathcal{X}}' = \sigma(b + \vec{\mathcal{X}} \dot{\theta} - \mathbf{1} \vec{\mathcal{X}}_{\max} \bar{\theta})$  where  $\vec{\mathcal{X}}_{\max}^{(\ell)} \in \mathbb{R}^{K^{(\ell-1)}}$  is the row vector of max-values over the rows of  $\vec{\mathcal{X}}$  and  $b$  is an additional bias parameter. In practice we can marginally increase generalization performance by factoring the parameters and using

$$\vec{\mathcal{X}}' = \sigma(b + (\vec{\mathcal{X}} - \mathbf{1} \vec{\mathcal{X}}_{\max}) \theta). \quad (4)$$

With multiple input/output channels, the complexity of this layer for each instance is  $\mathcal{O}(N K K')$ . Subtracting the mean or max over the set also reduces the internal covariate shift (Ioffe & Szegedy, 2015) and we observe that for deep networks (even using Tanh activation), batch-normalization is not required.

### 5.2 SET ANOMALY DETECTION

The objective here is for the deep model to find the anomalous face in each set, simply by observing examples and without any access to the attribute values. CelebA dataset (Liu et al., 2015) contains 202,599 face images, each annotated with 40 boolean attributes. We use  $64 \times 64$  stamps and using these attributes we build 18,000 sets, each containing  $N = 16$  images (on the training set) as follows: after randomly selecting two attributes, we draw 15 images where those attributes are present and a single image where both attributes are absent. Using a similar procedure we build sets on the test images. No individual person’s face appears in both train and test sets.

<sup>2</sup> $\hat{\oplus}$  can be different from  $\oplus$  in the set layer. This does not alter the invariance of  $f$  to  $\mathbb{S}^{\text{set}}$ .

Table 1: Classification accuracy and the (size of) representation used by different methods on the ModelNet40 dataset.

model	instance size	representation	accuracy
<b>set-convolution</b> + transformation (ours)	<b>5000</b> $\times$ <b>3</b>	point-cloud	90 $\pm$ .3%
<b>set-convolution</b> (ours)	<b>1000</b> $\times$ <b>3</b>	point-cloud	87 $\pm$ 1%
<b>set-convolution</b> (ours)	<b>100</b> $\times$ <b>3</b>	point-cloud	82 $\pm$ 2%
KNN graph-convolution (ours)	1000 $\times$ (3 + 8)	directed 8-regular graph	58 $\pm$ 2%
3DShapeNets (Wu et al., 2015)	30 <sup>3</sup>	voxels (using convolutional deep belief net)	77%
DeepPano (Shi et al., 2015)	64 $\times$ 160	panoramic image (2D CNN + angle-pooling)	77.64%
VoxNet (Maturana & Scherer, 2015)	32 <sup>3</sup>	voxels (voxels from point-cloud + 3D CNN)	83.10%
MVCNN (Su et al., 2015)	164 $\times$ 164 $\times$ 12	multi-view images (2D CNN + view-pooling)	90.1%
VRN Ensemble (Brock et al., 2016)	32 <sup>3</sup>	voxels (3D CNN, variational autoencoder)	95.54%
3D GAN (Wu et al., 2016)	64 <sup>3</sup>	voxels (3D CNN, generative adversarial training)	83.3%

Our deep neural network consists of 9 2D-convolution and max-pooling layers followed by 3 set-invariant layers (of the type given by Eq. (4)) and a softmax layer that assigns a probability value to each set member (Note that one could identify arbitrary number of outliers using a sigmoid activation at the output.) In the initial convolution and pooling layers we use the trick of Section 4.1, ignoring the set structure. Our trained algorithm successfully finds the anomalous face in 75% of test sets. Visually inspecting these instances suggests that the task is non-trivial even for humans; see Fig. 3. For details of the network model and more identification examples see Appendix D.

### 5.3 POINT CLOUD CLASSIFICATION

A low-dimensional point-cloud is a set of low-dimensional vectors. This type of data is frequently encountered in various applications from robotics and vision to cosmology. In these applications point-cloud data is often converted to voxel or mesh representation at a preprocessing step (e.g., Maturana & Scherer, 2015; Ravanbakhsh et al., 2016; Lin et al., 2004). Since the output of many range sensors such as LiDAR – which are extensively used in applications such as autonomous vehicles – is in the form of point-cloud, direct application of deep learning methods to point-cloud is highly desirable. Moreover, when working with point-clouds rather than voxelized 3D objects, it is easy to apply transformations such as rotation and translation as differentiable layers, and achieve this type of continuous invariances, that our framework cannot formulate.

Here, we show that treating the point-cloud data as a set, we can use the set-invariant layer to classify point-cloud representation of a subset of ShapeNet objects (Chang et al., 2015), called ModelNet40 (Wu et al., 2015). This subset consists of 3D representation of 9,843 training and 2,468 test instances belonging to 40 classes of objects; see Fig. 4. We produce point-clouds with 1000 particles each ( $x, y, z$ -coordinates) from the mesh representation of objects using the point-cloud-library’s sampling routine (Rusu & Cousins, 2011). Each set is normalized by the initial layer of the deep network to have zero mean and unit variance. Additionally we experiment with the K-nearest neighbor graph of each point-cloud and report the results using graph-convolution (see Appendix D for model details.)

Table 1 compares our method against the competition. Note that we achieve our accuracy using  $5000 \times 3$  dimensional representation of each object, which is much smaller than most other methods. All other techniques use either voxelization or multiple view of the 3D object for classification.<sup>3</sup> We see that reducing the number of particles to only 100 still produces comparatively good results. Using graph-convolution is computationally more challenging and produces inferior results in this setting. The results using 5000 particles is also invariant to small changes in scale and rotation around the  $z$ -axis (see Appendix D for details).

**Features.** To visualize the set-invariant layers, we used Adamax (Kingma & Ba, 2014) to locate 1000 particle coordinates maximizing the activation of each unit.<sup>4</sup> Activating the tanh units beyond

<sup>3</sup>The error-bar on our results is due to variations depending on the choice of particles during test time and it is estimated over three trials.

<sup>4</sup>We started from uniformly distributed set of particles and used a learning rate of .01 for Adamax, with first and second order moment of .1 and .9 respectively. We optimized the input in  $10^5$  iterations. The results of

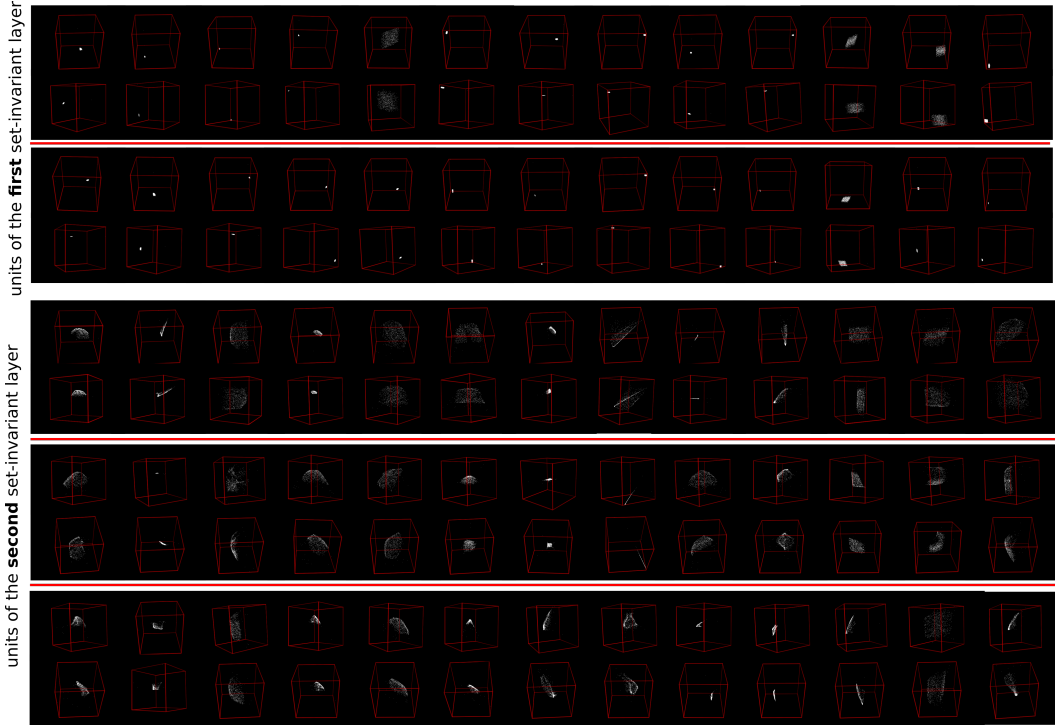


Figure 5: Each box is the particle-cloud maximizing the activation of a unit at the first (**top**) and second (**bottom**) set-invariant layers of our model.

the second layer proved to be difficult. Figure 5 shows the particle-cloud-features learned at the first and second layers of our deep network. We observed that the first layer learns simple localized (often cubic) point-clouds at different  $(x, y, z)$  locations, while the second layer learns more complex surfaces with different scales and orientations.

## 6 RELATED WORKS AND DISCUSSION

Several works have considered different approaches to deep-learning on graphs in the past (e.g., [Duvinaud et al., 2015](#); [Atwood & Towsley, 2015](#)). Graph convolution using the spectrum of graph was initially proposed by [Bruna et al. \(2013\)](#) and further developed in ([Defferrard et al., 2016](#); [Kipf & Welling, 2016](#)).

Our discussion of parameter-sharing across layers in acyclic structures (in Appendix B) is closely related to recursive neural networks (e.g., [Socher et al., 2013; 2011](#); [Irsoy & Cardie, 2014](#)) that use back-propagation through structure ([Goller & Kuchler, 1996](#)). [Sperduti & Starita \(1997\)](#) apply similar techniques for structure classification, where a topological ordering in cyclic structures is used to construct a directed acyclic graph.

Several works have studied invariance (and equivariance) in deep models using harmonic analysis and group theory. Scattering convolution networks ([Bruna & Mallat, 2013](#); [Sifre & Mallat, 2013](#)), group-convolution ([Christopher, 2014](#); [Cohen & Welling, 2016](#)) and symmetry networks ([Gens & Domingos, 2014](#)) are some examples that provide a general treatment of the topic. However, to our knowledge, these techniques do not directly extend to discrete structures such as graphs (unless the graph is a lattice). Other works in this area explicitly address known invariances through application of transformations that do not vary the output (e.g., [Jaderberg et al., 2015](#); [Dieleman et al., 2015](#)).

Fig. 5 are limited to instances where tanh units were successfully activated. Since the input at the first layer of our deep network is normalized to have a zero mean and unit standard deviation, we do not need to constrain the input while maximizing unit’s activation.



Contrasting our formulation with previous work on structure and symmetry in neural networks suggests several benefits and a drawback: Our approach is minimal in its use of machinery, yet quite general in application, and the notion of structure as defined here, has enough capacity to accommodate complex composite models. Moreover, any structure directly translates to a parameter-sharing scheme which has the benefit of being fast and easy to implement. However, this approach is also limited by the invariances achievable through parameter-sharing.

In the future, we would like to apply parameter-sharing schemes discussed here to real-word composite structures. Extending these ideas to deep generative models is also a direction we would like to explore in the future.

## REFERENCES

- Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- James Atwood and Don Towsley. Diffusion-convolutional neural networks. *arXiv preprint arXiv:1511.02136*, 2015.
- James Binney and Michael Merrifield. *Galactic astronomy*. Princeton University Press, 1998.
- Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016.
- Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.
- Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.
- Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.
- Olah Christopher. Groups and group convolutions. <http://colah.github.io/posts/2014-12-Groups-Convolution/>, 2014.
- Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.
- Taco S Cohen and Max Welling. Group equivariant convolutional networks. *arXiv preprint arXiv:1602.07576*, 2016.
- Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.
- Sander Dieleman, Kyle W Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2015.
- David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.
- Robert Gens and Pedro M Domingos. Deep symmetry networks. In *Advances in neural information processing systems*, pp. 2537–2545, 2014.
- Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pp. 347–352. IEEE, 1996.
- Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- Ozan Irsoy and Claire Cardie. Deep recursive neural networks for compositionality in language. In *Advances in Neural Information Processing Systems*, pp. 2096–2104, 2014.

- Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pp. 2017–2025, 2015.
- Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- Hong-Wei Lin, Chiew-Lan Tai, and Guo-Jin Wang. A mesh reconstruction algorithm driven by an intrinsic property of a point cloud. *Computer-Aided Design*, 36(1):1–9, 2004.
- Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.
- Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 922–928. IEEE, 2015.
- M. Ntampaka, H. Trac, D. Sutherland, S. Fromenteau, B. Poczós, and J. Schneider. Dynamical mass measurements of contaminated galaxy clusters using machine learning. *The Astrophysical Journal*, 2016.
- J. Oliva, B. Poczós, T. Verstynen, A. Singh, J. Schneider, F. Yeh, and W. Tseng. Fusso: Functional shrinkage and selection operator. In *International Conference on AI and Statistics (AISTATS)*, 2014.
- B. Poczós, L. Xiong, D. Sutherland, and J. Schneider. Nonparametric kernel estimators for image classification. In *25th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- Siamak Ravanbakhsh, Junier Oliva, Sebastien Fromenteau, Layne C Price, Shirley Ho, Jeff Schneider, and Barnabás Póczós. Estimating cosmological parameters from the dark matter distribution. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.
- Soumya Ray, Stephen Scott, and Hendrik Blockeel. Multi-instance learning. In *Encyclopedia of Machine Learning*, pp. 701–710. Springer, 2011.
- Eduardo Rozo and Eli S Rykoff. redmapper ii: X-ray and sz performance benchmarks for the sdss catalog. *The Astrophysical Journal*, 783(2):80, 2014.
- Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.
- Baoguang Shi, Song Bai, Zhichao Zhou, and Xiang Bai. Deeppano: Deep panoramic representation for 3-d shape recognition. *IEEE Signal Processing Letters*, 22(12):2339–2343, 2015.
- Laurent Sifre and Stéphane Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1233–1240, 2013.
- Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 129–136, 2011.
- Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, pp. 1642. Citeseer, 2013.
- Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.
- Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 945–953, 2015.
- Z. Szabo, B. Sriperumbudur, B. Poczós, and A. Gretton. Learning theory for distribution regression. *Journal of Machine Learning Research*, 2016.

- A. Tallavajhula, A. Kelly, and B. Póczos. Nonparametric distribution regression applied to sensor modeling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-16)*, 2016.
- Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *arXiv preprint arXiv:1610.07584*, 2016.
- Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1912–1920, 2015.
- Zhi-Hua Zhou, Yu-Yin Sun, and Yu-Feng Li. Multi-instance learning by treating instances as non-iid samples. In *Proceedings of the 26th annual international conference on machine learning*, pp. 1249–1256. ACM, 2009.

## Appendix

### A PROOFS

*Proof.* of proposition 2.1

For this it is enough to show that for a  $\mathbb{I}'$  in  $\mathbb{S}$ ,  $f_\theta(\pi(\vec{x}_{\mathbb{S}})) = f_\theta(\vec{x}_{\mathbb{S}})$ , when  $\pi \in \mathbb{S}_{\mathbb{I}'}$ .

$$\begin{aligned}
 f_\theta(\vec{x}_{\mathbb{S}}) &= \sigma \left( \bigoplus_{\mathbb{I} \in \mathbb{S}, x \in x_{\mathbb{I}}} \theta_{\mathbb{I}} x \right) = \\
 &= \sigma \left( \left( \bigoplus_{\mathbb{I} \in \mathbb{S}, \mathbb{I} \neq \mathbb{I}', x \in x_{\mathbb{I}}} \theta_{\mathbb{I}} x \right) \oplus \bigoplus_{x \in x_{\mathbb{I}'}} \theta_{\mathbb{I}'} x \right) = \\
 &= \sigma \left( \left( \bigoplus_{\mathbb{I} \in \mathbb{S}, \mathbb{I} \neq \mathbb{I}', x \in x_{\mathbb{I}}} \theta_{\mathbb{I}} x \right) \oplus \bigoplus_{x \in \pi(x_{\mathbb{I}'})} \theta_{\mathbb{I}'} x \right) = \\
 &= \sigma \left( \bigoplus_{\mathbb{I} \in \mathbb{S}, x \in \pi(x_{\mathbb{I}})} \theta_{\mathbb{I}} x \right) = \\
 &= f_\theta(\pi(\vec{x}_{\mathbb{S}}))
 \end{aligned}$$

□

*Proof.* of claim 4.2

For both Cardinal and Cartesian products we show that give  $\mathbb{S}_{\mathcal{X}}^2$  may be constructed from  $\mathbb{S}_{\mathcal{X}}^1$  using the operations above, we can also produce the product structure  $\mathbb{S}_{\mathcal{X}}^1 \times \mathbb{S}_{\mathcal{X}'}$  or  $\mathbb{S}_{\mathcal{X}}^1 \square \mathbb{S}_{\mathcal{X}'}$  using the same two operations.

Let us start with the *Cardinal product*:

$$\mathbb{S}_{\mathcal{X}^1}^1 \times \mathbb{S}_{\mathcal{X}'} = \{\mathbb{I}_{\mathcal{X}^1} \times \mathbb{I}_{\mathcal{X}'} \mid \mathbb{I}_{\mathcal{X}^1} \in \mathbb{S}_{\mathcal{X}^1}^1 \wedge \mathbb{I}_{\mathcal{X}'} \in \mathbb{S}_{\mathcal{X}'}\}.$$

where  $\mathbb{S}_{\mathcal{X}'}$  is in common on both sides of the rhs of Eq. (2) and therefore  $\mathbb{I}_{\mathcal{X}'}$  is in common as well. However, our assumption is that  $\mathbb{I}_{\mathcal{X}}^2 \in \mathbb{S}_{\mathcal{X}}^2$  is constructed from  $\mathbb{I}^1 \in \mathbb{S}_{\mathcal{X}}^1$  using by merge and shrinkage. We replace the merge and shrink operations for relations to their product form

1. Merging of products  $\mathbb{I}_{\mathcal{X}}^1 \times \mathbb{I}_{\mathcal{X}'}^2, \mathbb{I}_{\mathcal{X}}^2 \times \mathbb{I}_{\mathcal{X}'} \Rightarrow (\mathbb{I}_{\mathcal{X}}^1 \cup \mathbb{I}_{\mathcal{X}}^2) \times \mathbb{I}_{\mathcal{X}'}$
2. Shrinking of a product of relations  $\mathbb{I}_{\mathcal{X}}^1 \times \mathbb{I}_{\mathcal{X}'} \Rightarrow (\mathbb{I}_{\mathcal{X}}^1 \times \mathbb{I}_{\mathcal{X}'}) \setminus (\mathbb{I}'_{\mathcal{X}} \times \mathbb{I}_{\mathcal{X}'})$  for some  $\mathbb{I}'_{\mathcal{X}} \subseteq \mathbb{I}_{\mathcal{X}}^1$ .

By replacing the merge and shrink operations that derive  $\mathbb{S}_{\mathcal{X}}^2$  from  $\mathbb{S}_{\mathcal{X}}^1$ , we are guaranteed to obtain  $\mathbb{S}_{\mathcal{X}}^2 \times \mathbb{S}_{\mathcal{X}'}$  starting from  $\mathbb{S}_{\mathcal{X}}^1 \times \mathbb{S}_{\mathcal{X}'}$ .

For *Cartesian product*

$$\mathbb{S}_{\mathcal{X}} \square \mathbb{S}_{\mathcal{X}'} \stackrel{\text{def}}{=} \{\mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'} \mid \mathbb{I}_{\mathcal{X}} \in \mathbb{S}_{\mathcal{X}}\} \cup \{\mathbb{I}_{\mathcal{X}' \rightarrow \mathcal{X} \times \mathcal{X}'} \mid \mathbb{I}_{\mathcal{X}'} \in \mathbb{S}_{\mathcal{X}'}\}$$

Note that the second term  $\{\mathbb{I}_{\mathcal{X}' \rightarrow \mathcal{X} \times \mathcal{X}'} \mid \mathbb{I}_{\mathcal{X}'} \in \mathbb{S}_{\mathcal{X}'}\}$  is identical in both  $\mathbb{S}_{\mathcal{X}}^1 \square \mathbb{S}_{\mathcal{X}'}$  and  $\mathbb{S}_{\mathcal{X}}^2 \square \mathbb{S}_{\mathcal{X}'}$ . Therefore, we only need to show that

$$\{\mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}^1 \mid \mathbb{I}_{\mathcal{X}}^1 \in \mathbb{S}_{\mathcal{X}}^1\} \leq \{\mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}^2 \mid \mathbb{I}_{\mathcal{X}}^2 \in \mathbb{S}_{\mathcal{X}}^2\}.$$

Recall the definition of extension:

$$\mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'} \stackrel{\text{def}}{=} \{(x_{i,i'}, x_{j,i'}) \mid (x_i, x_j) \in \mathbb{I}_{\mathcal{X}} \wedge x_{i'} \in \mathcal{X}'\}.$$

Our assumption is that  $\mathbb{I}_{\mathcal{X}}^2 \in \mathbb{S}_{\mathcal{X}}^2$  is constructed from  $\mathbb{I}^1 \in \mathbb{S}_{\mathcal{X}}^1$  using by merge and shrink operations. We replace the merge and shrink with those operations applied to the extension of relations wrt  $\mathcal{X}'$  as follows

1. Merging of extensions  $\mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}^1, \mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}^2 \Rightarrow \mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}^1 \cup \mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}^2$
2. Expansion of extensions  $\mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}^1 \Rightarrow (\mathbb{I}_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}^1 \cup (\mathbb{I}'_{\mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}'}))$ .

By replacing the merge and shrink operations that derive  $\mathbb{S}_{\mathcal{X}}^2$  from  $\mathbb{S}_{\mathcal{X}}^1$ , we are guaranteed to obtain  $\mathbb{S}_{\mathcal{X}}^2 \square \mathbb{S}_{\mathcal{X}'}$  starting from  $\mathbb{S}_{\mathcal{X}}^1 \square \mathbb{S}_{\mathcal{X}'}$ . □

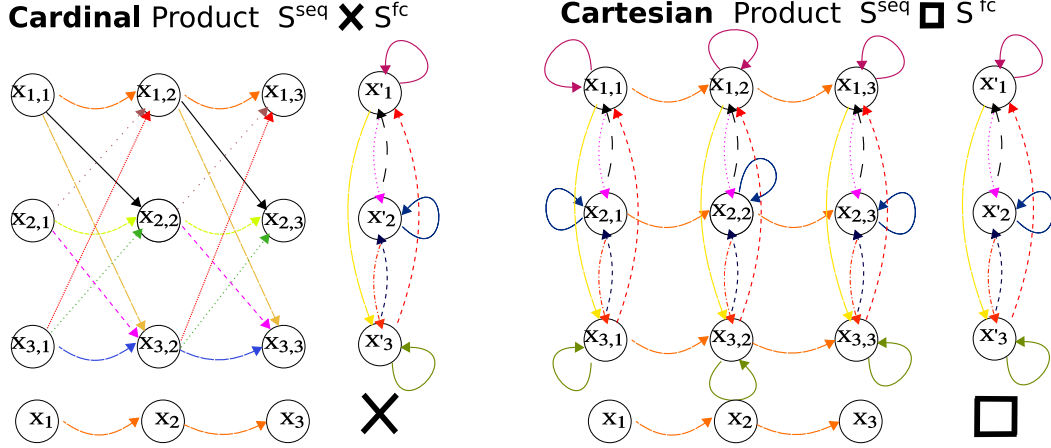


Figure 6: .

## B ACYCLIC STRUCTURE AND ANCESTRAL SAMPLING

Let  $\mathcal{G}_{\mathbb{S}} = (\mathcal{X}, \bigcup_{\mathbb{I} \in \mathbb{S}} \mathbb{I})$  be the directed graph associated with  $\mathbb{S}$  that merges all its relations. When  $\mathcal{G}_{\mathbb{S}}$  has no directed cycles (*i.e.*, it is a DAG), we call the structure  $\mathbb{S}$  an Acyclic Structure. This structure consistently identifies the inter-dependence of variables. Consistency is in the sense that the value of  $x \in \mathcal{X}$  does not implicitly depend on itself through a loop, and starting from the leaves, one could sample the value of each node once all its children have been assigned a value. This procedure is called ancestral sampling. Here, we show that with a simple inter-layer parameter-sharing scheme, a deep model implements ancestral sampling in a deep directed graphical model.

If for a given  $\mathbb{S}$ :

1.  $\mathcal{G}_{\mathbb{S}}$  is a DAG of depth  $D$
2. The number of layers  $L$  (in  $f_{\theta_1} \circ \dots \circ f_{\theta_L}$ ), is not smaller than  $D$
3. Layers share the same structure  $\mathbb{S}$  (up to an isomorphism of layers  $\mathcal{X}^{(\ell)} \simeq \mathcal{X}^{(\ell')} \quad \forall \ell, \ell'$ ) and parameters  $\theta = \theta_1 = \theta_2 = \dots = \theta_L$

then a forward pass through the network is equivalent to ancestral sampling.

To see why this is true let us follow a forward pass through this network, where at each iteration  $t$ , we calculate the output of functions at layer  $\ell = t$  and update their (deterministic) sample value  $x^{(t)} = f_{\theta_t}^{(\ell)}(x_{\mathbb{S}}^{(t-1)})$ . Since the leaf nodes do not depend on any other node their value does not change  $x^{(t)} = x^{(0)} = x \quad \forall t$ . Similarly the parents of these leaf nodes are fixed at the second level –*i.e.*, for these nodes  $x^{(t)} = x^{(1)} \quad \forall t \geq 1$ – and so on. The value of the root nodes is set at layer  $D$  and does not change from here on. This also suggests that for saving computation, we could sample a node at level  $h$  of the DAG, only at layer  $\ell = h = t$ . We call this computation scheme in a multi-layer perceptron that satisfies the conditions above, *minimal ancestral sampling*. If the structure does not reduce to a DAG, we could still use parameter-sharing between layers, to assimilate  $L$  iterations of Gibbs sampling.

**Example B.1.** *Vanilla recurrent neural network, effectively performs minimal ancestral sampling, where the structure is the Cardinal product of a directed sequence structure  $\mathbb{S}^{\text{seq}} = \{(x_n, x_{n+1}) \mid 1 \leq n \leq N_1\}$  and a fully-connected structure  $\mathbb{S}^{\text{full}} = \{(x_n, x_{n'}) \mid n, n' \in \{1, \dots, N_2\}\}$ . Figure 6 shows two different products of  $\mathbb{S}^{\text{seq}} \times \mathbb{S}^{\text{full}}$  and  $\mathbb{S}^{\text{seq}} \square \mathbb{S}^{\text{full}}$ , it is not hard to notice that the Cartesian product structure is not an acyclic structure.*



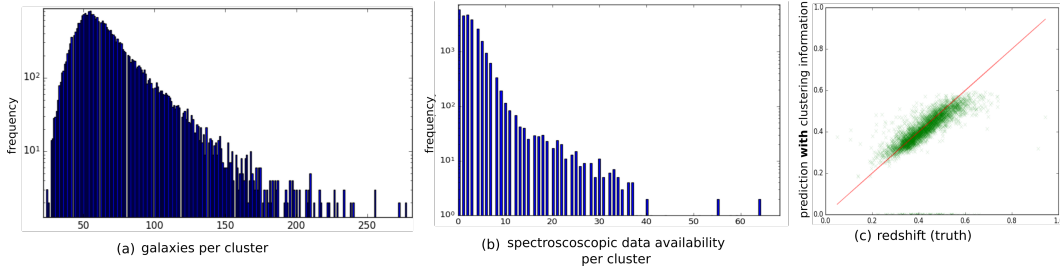


Figure 7: application of set-invariant layer to semi-supervised red-shift prediction using clustering information: **a)** distribution of cluster (set) size; **b)** distribution of reliable redshift estimates per cluster; **c)** prediction of redshift on test-set (versus ground-truth) using clustering information.

## C SEMI-SUPERVISED LEARNING WITH CLUSTERING INFORMATION

In this section we treat galaxy clusters as sets and use set-invariant layers to produce estimates of the galaxy red-shift (distance) using the observation of u,g,r, i and z photometric bands (Binney & Merrifield, 1998), measurements error bars, location in the sky and the probability of being the cluster center (for a total of 17 features per galaxy). This information is provided by redMaPPer galaxy cluster catalog (Roze & Rykoff, 2014) which contains photometric readings for 26,111 red galaxy clusters. Each cluster in this catalog has between  $\sim 20 - 300$  galaxies; see Fig. 7(a) for distribution of cluster sizes.

Having access to accurate spectroscopic red-shift estimates for a subset of these galaxies, we would like to use clustering information to produce better estimates of the red-shift using photometric data. Fig. 7(b) reports the distribution of available spectroscopic red-shift estimates per cluster.

We randomly split the data into 90% training and 10% test instances (of clusters) and define use the following architecture for semi-supervised learning using available spectroscopic redshift estimates and cluster information.

For this we use four set-invariant layers with 128, 128, 128 and 1 output channels respectively, where the output of the last layer is used as red-shift estimate. The squared loss of the prediction for available spectroscopic readings is minimized using mini-batches of size 128 with Adam. All layers except for the last layer use Tanh units and (simultaneous) dropout with 50% dropout rate. Fig. 7(c) shows the agreement of predictions with spectroscopic readings on the test-set. The average absolute error of the red-shift estimates is .06 in this setting.

We repeat this experiment, replacing the set layers with fully connected layers and only using the individual galaxies with available spectroscopic estimate for training. This achieves a mean absolute error of .08, suggesting that using clustering information indeed improves the performance of redshift estimation.

## D DETAILS OF MODELS

In the following, all our implementations use Tensorflow (Abadi et al., 2016).

### D.1 FACE OUTLIER DETECTION MODEL

Our model has 9 convolution layers with 3x3 kernels. The model has convolution layers with 32, 32, 64 feature-maps followed by max-pooling followed by 2D convolution layers with 64, 64, 128 feature-maps followed by another max-pooling layer. The final set of convolution layers have 128, 128, 256 feature-maps, followed by a max-pooling layer with pool-size of 5 that reduces the output dimension to batch-size  $N \times 256$ , where the set-size  $N = 16$ . This is then forwarded to three set-invariant layers with 256, 128 and 1 output channels. The output of final layer is fed to the Softmax, to identify the outlier. We use exponential linear units (Clevert et al., 2015), drop out with 20% dropout rate at convolutional layers and 50% dropout rate at the first two set layers.



Figure 8: Each row of the images shows a set, constructed from CelebA dataset images, such that all set members except for an outlier, share at least two attributes. The **outlier is identified with a red frame**. The model is trained by observing examples of sets and their anomalous members and **without access to the attributes**. The probability assigned to each member by the outlier detection network is visualized using a **red bar** at the bottom of each image. The probabilities in each row sum to one.

When applied to set layers, the selected feature (channel) is simultaneously dropped in all the set members of that particular set. We use Adam (Kingma & Ba, 2014) for optimization and use batch-normalization only in the convolutional layers. We use mini-batches of 8 sets, for a total of 128 images per batch.

## D.2 MODELS FOR POINT-CLOUDS CLASSIFICATION

**Set convolution.** We use a network comprising of 3 set-invariant layers with 256 channels followed by max-pooling over the set structure. The resulting vector representation of the set is then fed to a fully connected layer with 256 units followed by a 40-way softmax unit. We use Tanh activation at all layers and dropout on the layers after set-max-pooling (*i.e.*, two dropout operations) with 50% dropout rate. Applying dropout to set-invariant layers for point-cloud data deteriorated the performance. We observed that using different types of set-invariant layers (see Section 5.1) and as few as 64 channels for set-invariant layers changes the result by less than 5% in classification accuracy.

For the setting with 5000 particles, we increase the number of units to 512 in all layers and randomly rotate the input around the  $z$ -axis. We also randomly scale the point-cloud by  $s \sim \mathcal{U}(.8, 1./8)$ . For this setting only, we use Adamax (Kingma & Ba, 2014) instead of Adam and reduce learning rate from .001 to .0005.

**Graph convolution.** For each point-cloud instance with 1000 particles, we build a sparse K-nearest neighbor graph and use the three point coordinates as input features. We normalized all graphs at the

preprocessing step. For direct comparison with set-invariant layer, we use the exact architecture of 3 graph-convolution layer followed by set-pooling (global graph pooling) and dense layer with 256 units. We use exponential linear activation function instead of Tanh as it performs better for graphs. Due to over-fitting, we use a heavy dropout of 50% after graph-convolution and dense layers. Similar to dropout for sets, all the randomly selected features are simultaneously dropped across the graph nodes. We use a mini-batch size of 64 and Adam for optimization where the learning rate is .001 (the same as that of set-invariant counter-part).

Despite our efficient sparse implementation using Tensorflow, graph-convolution is significantly slower than set-convolution. This prevented a thorough search for hyper-parameters and it is quite possible that better hyper-parameter tuning would improve the results that we report here.